Homework 8 Introduction to Big Data Systems

Christoffer Brevik

November 17, 2024

1 Code overview

To solve this weeks assignment I made multiple files with specific uses. Below is a quick overview of these files and their uses:

- **partition.py:** My solution to Task1. This utilizes the Edge-cut and Vertex-cut to partition a graph.
- **partition_visualizer.py:** An alternative version of `partitionr.py` where we only focus on the `small-5.graph`, noting all edges and vertices in each partition, and graphing the results. This is the code used to visualize my results in Task1.
- **graph_maker.py:** This python script takes a list of edges and creates a graph in the binary format described in the assignment. This script exists as the sample graph from the `ReadMe`, which I could use to test my algorithm against, wasn't provided as a dataset. It already contains the edges for the sample graph.

1.1 How to Run my the code

1.1.1 Dependencies

For this assignment I was instructed to download `networkx` and `matplotlib` on the server node and inform about the dependencies installed in the report. If these are not instealled when you try to run the code, you can enter the following in the terminal at any location in the `./2024403421/` folder to install all of them:

```
python3 -m pip install networkx
python3 -m pip install matplotlib
```

1.1.2 Running the Script

Running my Partitioning program can be done by going into the `./2024403421/hw8_src` folder in the computational node, or the root folder of my delivery file. To run my program you use:

```
python3 partition.py [FILE] [NUM_PARTITIONS] [VISUALIZE_PARTITIONS]
```

Where:

- [**FILE**] is the location of our graph file. As I've already added shortcuts to the local graphs, you only need to enter the file name, given that it is in the same folder. *(Default value is small-5.graph)*
- [**NUM_PARTITIONS**] is how many partitions you want as a number. *(Default value is 3)*
- [**SHOW_DETAILS**] is True if you want the program to plot both the full graph as well as all partitions, as False if you don't want it to plot anything at all. **Should be disabled on all graphs except small-5.graph**. *(Default value is False)*

These values can also be left empty and the program automatically chooses the *small-5* graph with 3 partitions and with plotting graphs disabled.

Below I will describe the `partition.py` script that this commands runs, explain how the code works, and show what results you should expect from it

2 My code

To solve the task I decided to use Python. My code resides in the file `partition.py`, but since this file is 100 lines of code I will not show it in it's entirety. Rather I will display the most central funcitons that are used during the partitioning:

2.1 read_graph

This section works very similarly to the already exising example code, where it reads the edges from the specified file and saves them in a list format:

```
def read_graph(self):
    edges = []
    with open(self.file_path, "rb") as file:
        data = file.read(8)     # Read first 8 bytes
        while data:
            src, dst = struct.unpack("ii", data) # (4 byte src, 4 byte dst)
            edges.append((src, dst))
            data = file.read(8) # Read the next 8 bytes
    return edges
```

2.2 Edge-Cut Partition

```
def edge_cut_partition(self, edges):
    partitions = defaultdict(
        lambda: {
            'master_vertices': set(),
            'total_vertices': set(),
            'replicated_edges': 0,
            'edges': []
        }
    )
    vertex_to_partition = {}

    for src, dst in edges:
        # We hash the vertexes to assign them to partitions
        src_partition = vertex_to_partition.get(src, hash(src) % self.num_partitions)
        dst_partition = vertex_to_partition.get(dst, hash(dst) % self.num_partitions)

        # Assign the vertex to the partition
        vertex_to_partition[src] = src_partition
        vertex_to_partition[dst] = dst_partition

        # Assign edges and count replicated edges
        if src_partition != dst_partition:
            partitions[src_partition]['replicated_edges'] += 1
            partitions[dst_partition]['replicated_edges'] += 1

        partitions[src_partition]['edges'].append((src, dst))
        partitions[dst_partition]['edges'].append((src, dst))

        # Update master and total vertices
        partitions[src_partition]['master_vertices'].add(src)
        partitions[dst_partition]['master_vertices'].add(dst)
        partitions[src_partition]['total_vertices'].update([src, dst])
        partitions[dst_partition]['total_vertices'].update([src, dst])

    # Print partition statistics
    for i, partition in sorted(partitions.items()):
        print(f"Partition {i}")
        print(len(partition['master_vertices']))
        print(len(partition['total_vertices']))
        print(partition['replicated_edges'])
        print(len(partition['edges']))
```

```
        # Show the partitions side by side
        if self.show_details:
            self.show_graph(partitions, title="Edge-Cut Partitioning")
```

2.3 Vertex-Cut Partition

```
    def vertex_cut_partition(self, edges):
        partitions = defaultdict(
            lambda: {
                'master_vertices': set(),
                'total_vertices': set(),
                'edges': []
            }
        )
        vertex_partitions = defaultdict(set)

        for src, dst in self.read_graph():
            edge_partition = hash((src, dst)) % self.num_partitions
            partitions[edge_partition]['edges'].append((src, dst))
            vertex_partitions[src].add(edge_partition)
            vertex_partitions[dst].add(edge_partition)

        for vertex, assigned_partitions in vertex_partitions.items():
            master_partition = random.choice(list(assigned_partitions))
            for partition_id in assigned_partitions:
                partitions[partition_id]['total_vertices'].add(vertex)
                if partition_id == master_partition:
                    partitions[partition_id]['master_vertices'].add(vertex)

        for i, partition in sorted(partitions.items()):
            print(f"Partition {i}")
            print(f"{len(partition['master_vertices'])}")
            print(f"{len(partition['total_vertices'])}")
            print(f"{len(partition['edges'])}")

        if self.show_details:
            self.show_graph(partitions, title="Vertex-Cut Partitioning")
```

3 Explanation of Code

This section provides an overview of how the graph partitioning code works to divide edges and vertices among different partitions.

3.1 Initiating GraphPartitioner

After importing the relevant libraries, the `GraphPartitioner` object is initiated with the file path, the number of partitions (`num_partitions`), and an optional flag `show_details`, which controls whether or not we show the full and partitioned graphs.

The first step in the process is reading the graph from the binary file using the `read_graph` function. In this function, the graph edges are read from the file in 8-byte chunks, each containing two 4-byte integers representing the source (`src`) and destination (`dst`) vertices. These edges are stored in a list, which will be used for partitioning the graph later.

After reading the graph, the partitioning algorithms are run, starting with the Edge-Cut partitioning method. The partitioning results are printed to the console, and optionally, the graphs of the partitions are displayed using the `show_graph` function.

3.2 Edge-Cut Partitioning

The `edge_cut_partition` function is responsible for the edge-cut partitioning algorithm. This method partitions the graph based on the edges and tries to minimize the number of replicated edges across partitions. Here's how it works:

1. For each edge in the graph, the algorithm checks the partition of both the source (`src`) and destination (`dst`) vertices. If a vertex is not yet assigned to a partition, it is assigned based on the hash of the vertex's ID, ensuring an even distribution of vertices across partitions.
2. If the source and destination vertices belong to different partitions, the edge is considered replicated, and the count of replicated edges for both partitions is incremented.
3. The edge is added to the list of edges for both partitions.
4. The algorithm maintains two sets for each partition: `master_vertices` (the vertices that are the primary or "master" vertices of the partition) and `total_vertices` (all vertices that belong to the partition, including replicated ones).
5. The statistics for each partition, including the number of master vertices, total vertices, the number of replicated edges, and the number of edges, are printed to the console.
6. If `show_details` is set to `True`, the partitions are visualized using the `show_graph` method.

3.3 Vertex-Cut Partitioning

The `vertex_cut_partition` function implements a vertex-cut partitioning approach, where the focus is on minimizing the number of edges crossing partition boundaries. The process is as follows:

1. For each edge in the graph, the function hashes the edge (using the source and destination vertex pair) and delegates it to a partition based on this hash.
2. The edge is added to the list of edges for the corresponding partition.
3. The `vertex_partitions` dictionary keeps track of which partitions each vertex is assigned to. For each vertex, the set of partitions it is assigned to is stored.
4. For each vertex, one of its assigned partitions is randomly chosen as the `master_partition`. The selected partition is the one where the vertex will be the master, and the other partitions are considered as containing replicated copies of the vertex.
5. The function then updates the `master_vertices` and `total_vertices` sets for each partition accordingly.
6. After partitioning, statistics for each partition, such as the number of master vertices, total vertices, and the number of edges, are printed to the console.
7. If `show_details` is set to `True`, the partitions are visualized using the `show_graph` method.

3.4 Visualization of Partitions

Both partitioning methods optionally visualize the graph partitions using the `show_graph` function. This function generates subplots for each partition, where each subplot displays a directed graph with edges and vertices. Using the `networkx` library, we can automatically The vertices are arranged using a spring layout for a visually appealing representation. Here we color master vertices green, to show the distribution of these. This will be used to show my results when partitioning `small-5.graph` in Chapter 5.

4 Results

After discussing with the TA, I was reccomended to use all datasets on all of the stated partitions (2, 3, 4, 8). I Have therefore chosen to follow his advice:

4.1 roadNet-PA

4.1.1 2 partitions

```
Edge-Cut Partitioning:
Partition 0
544015
1034541
1836224
3084312
Partition 1
544077
1034699
1836224
3083280

Vertex-Cut Partitioning:
Partition 0
543514
1006204
1543145
Partition 1
544578
1005736
1540651
```

4.1.2 3 partitions

```
Edge-Cut Partitioning:
Partition 0
362698
903163
1580526
2056962
Partition 1
362677
903390
1578970
2054534
Partition 2
362717
903339
1578856
2056096

Vertex-Cut Partitioning:
Partition 0
362197
935015
1028823
Partition 1
363185
935137
1026921
Partition 2
362710
935495
1028052
```

4.1.3 4 partitions

```
Edge-Cut Partitioning:
Partition 0
272003
768472
1289464
1542240
Partition 1
272040
769210
1290760
1542116
Partition 2
272012
768605
1290676
1542072
Partition 3
272037
768232
1288516
1541164

Vertex-Cut Partitioning:
Partition 0
272033
830880
770168
Partition 1
272065
830908
770202
Partition 2
271879
831893
772977
Partition 3
272115
830809
770449
```

4.1.4 8 partitions

```
Edge-Cut Partitioning:
Partition 0
136004
456672
716976
771452
Partition 1
136010
456707
717520
771100
Partition 2
136003
455884
715800
770596
Partition 3
136001
455906
715324
770200
Partition 4
135999
456467
716552
770788
Partition 5
136030
456307
716596
771016
Partition 6
136009
456320
716644
771476
Partition 7
136036
456324
715936
770964

Vertex-Cut Partitioning:
Partition 0
136011
563056
385006
Partition 1
135909
563568
385270
Partition 2
136463
564361
386100
```

```
Partition 3
136179
563454
385391
Partition 4
136266
563650
385162
Partition 5
135674
563526
384932
Partition 6
136120
564993
386877
Partition 7
135470
563502
385058
```

4.2 synthesized-1b

4.2.1 2 partitions

```
Edge-Cut Partitioning:
Partition 0
500000
877662
2391386
4833868
Partition 1
500000
868729
2391386
4726860

Vertex-Cut Partitioning:
Partition 0
498932
745700
2389141
Partition 1
501068
746638
2391223
```

4.2.2 3 partitions

```
Edge-Cut Partitioning:
Partition 0
333334
710174
2062022
3000144
Partition 1
333333
737528
2132636
3210848
Partition 2
333333
755501
2179894
3349736

Vertex-Cut Partitioning:
Partition 0
332339
601350
1592327
Partition 1
334054
602168
1593687
Partition 2
333607
602129
1594350
```

4.2.3 4 partitions

```
Edge-Cut Partitioning:
Partition 0
250000
622244
1762086
2329252
Partition 1
250000
619763
1755857
2316339
Partition 2
250000
653409
1849832
2504616
Partition 3
250000
634887
1802989
2410521

Vertex-Cut Partitioning:
Partition 0
250386
509682
1195568
Partition 1
250265
510000
1196101
Partition 2
249038
508746
1193573
Partition 3
250311
509518
1195122
```

4.2.4 8 partitions

```
Edge-Cut Partitioning:
Partition 0
125000
407801
1036969
1182431
Partition 1
125000
391315
988563
1118921
Partition 2
125000
438279
1082516
1244154
Partition 3
125000
418170
1064240
1219202
Partition 4
125000
400742
1009353
1146821
Partition 5
125000
412761
1047520
1197418
Partition 6
125000
422689
1094686
1260462
Partition 7
125000
412181
1043331
1191319

Vertex-Cut Partitioning:
Partition 0
124930
328481
598402
Partition 1
125297
328315
598234
Partition 2
124886
327619
596978
```

```
Partition 3
125059
328423
596992
Partition 4
124671
328189
597166
Partition 5
125408
328776
597867
Partition 6
124847
327974
596595
Partition 7
124902
328214
598130
```

4.3 twitter-2010

```
My program sadly never finished as the computational node would crash due to the excess data made
```

5 Partitions in more deatil

I will now look into a tri-partition of *small-5.graph*. To aid me in this section of the assignment I created the `partition_visualizer.py`. It is a modified version `partition.py` where the actual nodes in each partition is displayed. There are also some miner changes, like removing the system arguments in favor of only using `small-5.graph` and always showing graphs. Otherwise it is completely unchanged, and due to using hashing when separating the vertices and edges, it gives the exact same result. Therefore both scripts returns the same values shown in 5.1, 5.2, but different console logs as displayed in 5.3.

I will therefore focus on `partition.py` for the first sub-chapters and show both scritps results in 5.3. When either of the python scripts is run with the small dataset we get the initial graph:

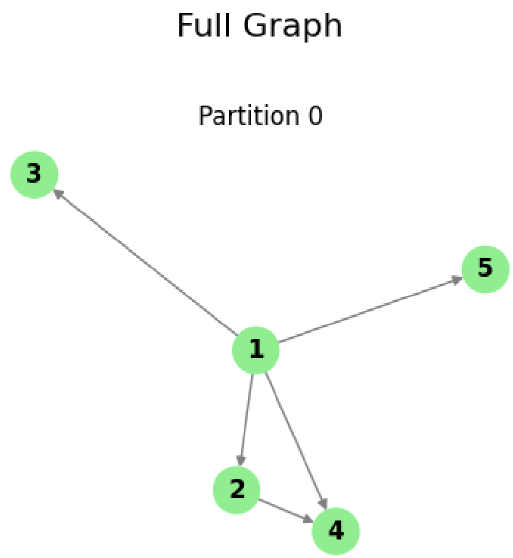

Figure 1: The small-5.graph that will be partitioned

5.1 Edge-Cut

After the program has finished the *edge_cut_partition*-function we get the following graphs:

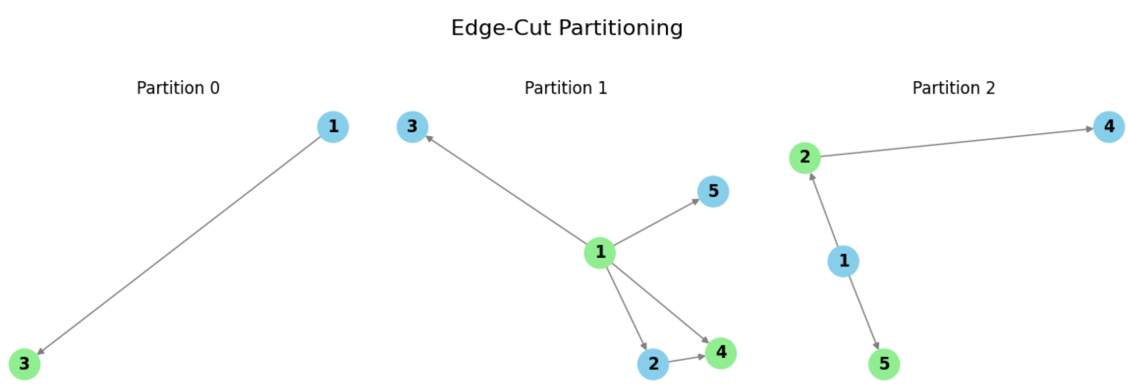

Figure 2: The small-5.graph after running Edge-Cut

Here we see that, while we expect a equal dispersal of master edges (colored green), we actually have an imbalance due to it only being 5 vertices and 3 partitions. Testing the same program on the sample graph provided in the `ReadMe`, which has 6 vertices, we get 2 vertices per partition, meaning that our program is indeed working. Still, as this algorithm focuses on Vertices, we see that the edges per partition is greatly disproportionate.

5.2 Vertex-Cut

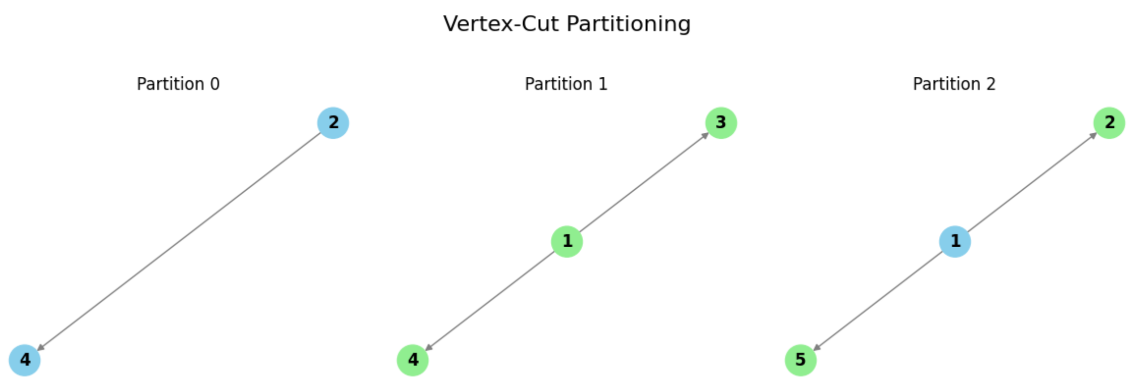

Figure 3: The small-5.graph that will be partitioned

Here we see that, like in the Edge-Cut we have a small inbalance in the edges per partition due to it not being divisible by 3. Still, as this algorithm focuses on edges, and master vertices are chosen at random, we see that the master vertices per partition can be greatly disproportionate.

5.3 Output of my program

Running my code on `partition.py` I get::

```
Edge-Cut Partitioning:
Partition 0
1
2
1
1
Partition 1
2
5
4
6
Partition 2
2
4
3
3

Vertex-Cut Partitioning:
Partition 0
1
2
1
Partition 1
1
3
2
Partition 2
3
```

```
3
2
```

While if I run the program on the modified version I get:

```
Edge-Cut Partitioning:
Partition 0
{3}
{1, 3}
1
[(1, 3)]
Partition 1
{1, 4}
{1, 2, 3, 4, 5}
4
[(1, 2), (1, 3), (1, 4), (1, 4), (1, 5), (2, 4)]
Partition 2
{2, 5}
{1, 2, 4, 5}
3
[(1, 2), (1, 5), (2, 4)]

Vertex-Cut Partitioning:
Partition 0
{2}
{2, 4}
[(2, 4)]
Partition 1
{1, 3, 4}
{1, 3, 4}
[(1, 3), (1, 4)]
Partition 2
{5}
{1, 2, 5}
[(1, 2), (1, 5)]
```

Both of these outputs matches our figures and we can therefore conclude that these are a correct representation of what my program outputs after both partitions.

6 Greedy Heuristic and Hybrid-Cut approach

To solve this task I implemented a Hybrid-Cut and Balanced p-way Hybrid-Cut partitioning algorithm. I will now explain how they work, and then run them on some of the datasets and show the results.

7 Explanation of Code

This section provides an overview of how my graph partitioning code works to divide edges and vertices among different partitions.

7.1 Hybrid-Cut Partitioning

The `hybrid_cut_partition` function is responsible for partitioning the graph using a hybrid approach based on PowerLyra. It divides the graph into partitions using both edge-cut and vertex-cut strategies, depending on the degree of the vertices.

1. The degree of each vertex is computed by iterating over all edges in the graph. This helps determine the strategy for each vertex.
2. Vertices with degrees higher than a threshold `theta` are assigned to partitions using an edge-cut strategy. The vertices are assigned to partitions based on the hash value of their identifiers.
3. For low-degree vertices, the algorithm uses a greedy heuristic to assign them to the partition with the least replication cost. Replication cost is calculated based on whether the vertices are already present in the partition's master vertex set.
4. Once vertices are assigned, edges are placed in the corresponding partitions. Each partition is ensured to have at least one master vertex, and the partition statistics are printed.
5. If `show_details` is enabled, the graph partitions are visualized using `show_graph`.

7.2 Greedy Vertex-Cut Partitioning

The `greedy_heuristic_partition` function is responsible for partitioning the graph using a greedy heuristic approach based on vertex degrees.

1. The degree of each vertex is calculated by iterating over all edges in the graph.
2. For each edge, the two vertices are assigned to partitions based on their degrees using the modulo operation `vertex_degrees[src] % self.num_partitions`. This ensures that vertices with similar degrees are placed in the same partition.
3. One of the two vertices from an edge is randomly selected as the master vertex for the partition, ensuring that each partition has a master vertex.
4. After assigning edges, the algorithm checks if any partitions are empty. If any partitions lack vertices, a random vertex is assigned to ensure that every partition contains at least one vertex.
5. The size of the master vertices, total vertices, and edges in each partition are printed. If `show_details` is enabled, the partitions are visualized.

8 Results

As this task did not state on how much I should test it for, I decided to run my code on roadNet-PA and synthesized-1b datasets for sizes 2, 3, 4 and 8. I've filtered out the irellevant functions so I got this

8.1 roadNet-PA

8.1.1 2 partitions

```
Hybrid-Cut Partitioning:
Partition 0
544184
1034541
3084310
Partition 1
543908
1034361
3082942

Greedy-Cut Partitioning:
Partition 0
1088092
1088092
6167592
Partition 1
1
1
0
```

8.1.2 3 partitions

```
Hybrid-Cut Partitioning:
Partition 0
362920
903267
2057066
Partition 1
362574
903186
2054328
Partition 2
362598
903103
2055858

Greedy-Cut Partitioning:
Partition 0
533095
533927
3211032
Partition 1
94841
98512
440758
Partition 2
419131
455653
2515802
```

8.1.3 4 partitions

```
Hybrid-Cut Partitioning:
Partition 0
272263
768654
1542420
Partition 1
271956
769042
1541948
Partition 2
271921
768423
1541890
Partition 3
271952
768062
1540994

Greedy-Cut Partitioning:
Partition 0
357355
359246
2516060
Partition 1
1
1
0
Partition 2
716174
728846
3651532
Partition 3
1
1
0
```

8.1.4 8 partitions

```
Hybrid-Cut Partitioning:
Partition 0
136303
456930
771710
Partition 1
135965
456617
771010
Partition 2
135961
455800
770512
Partition 3
135957
455818
770112
Partition 4
135960
456389
770710
Partition 5
135991
456229
770938
Partition 6
135960
456222
771378
Partition 7
135995
456242
770882

Greedy-Cut Partitioning:
Partition 0
267158
267269
2138256
Partition 1
1
1
0
Partition 2
149460
196080
454296
Partition 3
1
1
0
Partition 4
88232
91977
377804
```

```
Partition 5
1
1
0
Partition 6
531962
532766
3197236
Partition 7
1
1
0
```

8.2 synthesized-1b

8.2.1 2 partitions

```
Hybrid-Cut Partitioning:
Partition 0
517871
880341
4827193
Partition 1
492327
849617
4702865

Greedy-Cut Partitioning:
Partition 0
344395
429412
4876138
Partition 1
384734
570588
4684590
```

8.2.2 3 partitions

```
Hybrid-Cut Partitioning:
Partition 0
355766
718156
3000844
Partition 1
329123
721977
3195395
Partition 2
328736
739771
3333819

Greedy-Cut Partitioning:
Partition 0
168076
186037
3129612
Partition 1
286604
456212
3200300
Partition 2
286006
357751
3230816
```

8.2.3 4 partitions

```
Hybrid-Cut Partitioning:
Partition 0
274598
634150
2333836
Partition 1
247103
606530
2304708
Partition 2
247017
640511
2493026
Partition 3
246560
621543
2398488

Greedy-Cut Partitioning:
Partition 0
98800
103987
2413308
Partition 1
241027
408545
2280309
Partition 2
257604
325425
2462830
Partition 3
146795
162043
2404281
```

8.2.4 8 partitions

```
Hybrid-Cut Partitioning:
Partition 0
152453
427462
1192466
Partition 1
123773
383618
1113245
Partition 2
123886
431000
1238673
Partition 3
123693
410501
1213465
Partition 4
123673
393114
1141126
Partition 5
123481
404854
1191364
Partition 6
123440
414834
1254419
Partition 7
123383
404364
1185300

Greedy-Cut Partitioning:
Partition 0
25794
25860
1094320
Partition 1
188356
355204
1201788
Partition 2
218472
285656
1405992
Partition 3
116084
130825
1275603
Partition 4
73512
78127
1318988
```

```
Partition 5
52045
53341
1078521
Partition 6
39345
39769
1056838
Partition 7
31066
31218
1128678
```